Accurate Sequence Alignment using Distributed Filtering on GPU Clusters

Reza Farivar*
Shivaram Venkataraman§
Roy H. Campbell*

*University of Illinois at Urbana-Champaign
§University of California, Berkeley
Problem Introduction

• What is Gene Sequencing?

• 1st Phase, getting the reads

• 2nd Phase, Aligning them to a reference DNA

Thymine (Yellow) = T  Guanine (Green) = G
Adenine (Blue) = A  Cytosine (Red) = C
Problem Description

• Human genome sequence
  – 3.5 billion nucleotides
    • A C T G
  – Less than 30 chromosomes
  – Unidentified bases (N)

• Reads
  – Explore a sequence by designing and searching for short DNA reads

• Types of disagreements
  – Insertions, Deletions and Mismatches
Why is the Problem So Hard

• Divide reference DNA into **patterns** using a sliding window
• Brute Force - $(3 \times 10^9) \times (20 \times 10^6)$

Using Quad core, each core working on two datasets in parallel using SSE will take 195 days to complete!
Our Approach

PHASE 1
• Generate Masks for Filter Phase of the algorithm
• Reduce the number of Masks
• Filter the Pattern-Read pairs

PHASE 2
• Align them using an accelerator like GPUs, get rid of the False positives
Phase 1 - Masks

• Related to the q-gram lemma
  – Based on pigeon hole principle
• If $n$ “disagreements” are spread out in a string, and if we divide it in $n+k$ divisions, at least $k$ division will be an exact match
Distributed Filtering

• Using a sliding window, create patterns
• For each mask, make a “masked array” from all patterns
• For each read, mask with a proper value and search in the corresponding masked array
• If found, this pattern/read are a potential match for phase 2
Creating the distributed filter

Sorted Masked Arrays

1 1 0 1 0 1 0 1 1

CCTT00 CC00CA 00CATT
CTTC00 CT00AT 00CCTT
GCCT00 GC00TC 00CTTC
TGCC00 TG00TT 00TCAT
TTCA00 TT00TT 00TTCA

Distributed pigeon hole filter
Masked Read Matching

A Short Read

CCATCA

1 1 0

1 0 1

0 1 1

CCAT00

CC00CA

00ATCA

Sorted Masked Arrays

1 1 0

1 0 1

0 1 1

CCTT00

CC00CA

00CATT

CTTC00

CT00AT

00CCTT

GCCT00

GC00TC

00CTTC

TGCC00

TG00TT

00TCAT

TTCA00

TT00TT

00TTCA
## Performance of filtering

<table>
<thead>
<tr>
<th>Filter Passed %</th>
<th>Avg. No. of Patterns</th>
<th>Masks Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>85.38</td>
<td>13.6</td>
<td>1-1-1-2</td>
</tr>
<tr>
<td>13.39</td>
<td>21.38</td>
<td>1-1-1-3</td>
</tr>
<tr>
<td>2.84</td>
<td>46.66</td>
<td>1-1-1-4</td>
</tr>
</tbody>
</table>
Phase 2: Post Filter Matching

• Aligning two short sequences
  – Insertion, Deletion, Mismatch
  – Example:
    • X: kitten-
    • Y: sitting

• Needleman-Wunsch
  – Global Alignment
  – Create a matrix, with each element showing the score of the optimal alignment of two smaller subsequences. Then trace back.
Needleman Wunsch Algorithm

1. Fill the matrix (forward pass)
   - \( S(i,j) \) is the optimal alignment score for substrings \( X[0..i] \) and \( Y[0..j] \)
   - \( S(i,j) = f(S(i-1,j), S(i,j-1), S(i-1,j-1), X[i], Y[j]) \)

2. Trace-back (backward pass)
Limited Fast Memory

• NW in Shared Memory
  – Amount of shared memory available per streaming multiprocessor (in Tesla) = 16KB
  – For 32 long strings, 1024 bytes required → 16 threads per SM
  – For 128 long strings → 1 thread per SM!

Problem with 16 threads per SM
  – Very low GPU utilization
  – Unable to hide RAW latency
Matrix in Global Memory

- Allows hundreds of threads per streaming multiprocessor
- Many threads can be used to hide latency incurred by each thread, normal GPU practice

PROBLEM
- In the best case, we are global memory bound
- The algorithm does not support coalesced memory accesses during back-tracing
- This results in serializing the threads
Algorithm Walk through

1. Divide the matrix into quadrants, store the boundaries
2. Start from the bottom right quadrant $i,j = 31$
   - Fill the alignment matrix for the quadrant
   - Back trace
   - Decide on a new quadrant
3. Repeat 2 until $i,j = 0$
Storing The Boundaries

- Calculation of each quadrant requires the first row and the first column of the quadrant

- Storing in shared memory $\rightarrow$ only 2 warps per SM $\rightarrow$ 6% occupancy

- Storing in global memory $\rightarrow$ Shared memory usage 81 bytes $\rightarrow$
  Threads per SM=192 $\rightarrow$ 19% GPU utilization 😊

256 Bytes + 81 Bytes
Experimental Results: Speedup

- NVIDIA GTX 275
- Intel Core i7 920
Run Time Break Down

Run Time Details (in seconds)

- Read from Disk: 4 seconds
- Creating masked array: 12 seconds
- Binary search for all reads in one masked array: 9 seconds
- GPU processing time: 2 seconds
- Misc: 5 seconds
Future Work

• Scaling to public clouds
  – Distributed filtering using a cluster of GPUs
  – Integration with standard bioinformatics file formats

• Length of short sequences
  – Currently 32, we are extending to 100 and longer
Backup Slides
Goals and Assumptions

• Goals:
  – Make the kernel compute bound
    • Minimize global memory accesses, Ensure coalescing
  – Maximize use of fast memory
    • Shared memory, registers
  – Eliminate diverging flows

• Assumption:
  – If we eliminate memory bounds, we will have extra cycles to spare → re-compute when needed
  – Bank-conflict-free shared memory access is as fast as registers
The Proposed Algorithm

- Dynamically construct smaller quadrants while backtracing
- Solve each quadrant in shared memory

<table>
<thead>
<tr>
<th>A</th>
<th>T</th>
<th>C</th>
<th>C</th>
<th>T</th>
<th>A</th>
<th>G</th>
<th>G</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>T</td>
<td>C</td>
<td>C</td>
<td>T</td>
<td>A</td>
<td>G</td>
<td>G</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>T</td>
<td>G</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>G</td>
<td>T</td>
</tr>
<tr>
<td>A</td>
<td>T</td>
<td>G</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>G</td>
<td>T</td>
</tr>
<tr>
<td>A</td>
<td>T</td>
<td>G</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>G</td>
<td>T</td>
</tr>
</tbody>
</table>
Optimization with Registers

- 32 long strings \(\rightarrow\) boundaries take 256 bytes
- **Trick**: store the 256 bytes in 32 registers
- **Problem**: Registers are not array addressable
  - Need to use the registers conditionally
    \(\rightarrow\) divergent code 😞
- **Solution**: We use registers as **bulk** storage, paging into shared memory
- Shared memory usage **81 bytes** \(\rightarrow\)
  19% GPU utilization, no global memory
Runtime vs. Speedup

Normalized run time vs. Number of pairs to compare

- Blue line: Normalized OpenMP CPU time
- Red line: Normalized GPU time
Experimental Results

• Compared to a regular Needleman Wunsch implementation on an Intel i7 processor our implementation on GPUs is **22x faster**.

• Compared to a multithreaded implementation of the regular Needleman Wunsch using Open-MP on 4 Intel i7 cores with Hyper threading support (8 HT cores), our implementation is up to **8x faster**.

• What does this mean? Gene alignment for a human DNA with 3 Billion reads will now take 27 minutes instead of 3 hours!!